THE REPAST SIMPHONY DEVELOPMENT ENVIRONMENT

M.J. NORTH,[*] Argonne National Laboratory, Argonne, IL,
and The University of Chicago, Chicago, IL
T.R. HOWE, Argonne National Laboratory, Argonne, IL
N.T. COLLIER, Argonne National Laboratory, Argonne, IL,
and PantaRei Corp., Cambridge, MA
J.R. VOS, Argonne National Laboratory, Argonne, IL,
and the University of Illinois at Urbana-Champaign, Urbana, IL

ABSTRACT

Repast is a widely used free and open-source agent-based modeling and simulation toolkit. Three Repast platforms are currently available, each of which has the same core features but a different environment for these features. Repast Simphony (Repast S) extends the Repast portfolio by offering a new approach to simulation development and execution. The Repast S development environment is expected to include advanced features for agent behavioral specification and dynamic model self-assembly. This paper introduces the architecture and core features of the Repast S development environment. A related paper in the *Agent 2005* conference proceedings by the same authors that is titled "Repast Simphony Runtime System" discusses the Repast S model execution environment.

Keywords: Agent-based modeling and simulation, Repast, toolkits, development environments

INTRODUCTION

Repast is a widely used free and open-source agent-based modeling and simulation toolkit (ROAD 2005; North et al. 2006). Three Repast platforms are currently available, each of which has the same core features but a different environment for these features.

Repast Simphony (Repast S) extends the Repast portfolio by offering a new approach to simulation development and execution. The Repast S development environment is expected to include advanced features for agent behavioral specification and dynamic model self-assembly. This paper introduces the architecture and core features of the Repast S development environment. A related paper in these conference proceedings (North et al. 2005) discusses the Repast S runtime environment.

It is important to note that Repast S and its related tools are still under development. This paper presents the most current information as of the time it was written. However, changes may occur before the planned final release.

[*] *Corresponding author address*: Michael J. North, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439; e-mail: north@anl.gov.

RELATED WORK

There are a variety of existing agent-based modeling toolkits. Repast J, Repast Py, Repast .NET, NetLogo, and Swarm are just a few examples (ROAD 2005; Wilensky 1999; SDG 2005). The growing agent-based modeling literature suggests that the existing toolkits have been useful for many modelers. However, as outlined in North et al. (2005), more is needed. In particular, with respect to development environments, there are continuing needs to reduce the distance between modelers and programmers, automate common tasks, and directly support model and enterprise information system integration. The Repast S development environment is explicitly intended to meet these needs.

THE REPAST S MODEL IMPLEMENTATION BUSINESS PROCESS

As discussed in North et al. (2005), the Repast S model implementation business process is as follows:

- The modeler creates model pieces, as needed, in the form of plain old Java objects (POJOs), often using automated tools.

- The modeler uses declarative configuration settings to pass the model pieces and legacy software connections to the Repast S runtime system.

- The modeler uses the Repast S runtime system to declaratively tell Repast S how to instantiate and connect model components.

- Repast S automatically manages the model pieces based on (1) interactive user input and (2) declarative or imperative requests from the components themselves.

The POJO model components can represent anything, but are most commonly used to represent the agents in the model. The POJOs can be created using any method. This paper discusses one powerful way to create POJOs for Repast S, namely, the Repast Simphony development environment. However, any method from hand coding, to wrapping binary legacy models, to connecting into enterprise information systems can be used to create the Repast S POJO model components.

Regardless of the source of the POJOs, the Repast S runtime system is used to configure and execute Repast S models. North et al. (2005) details the Repast S runtime system. In summary, the Repast S runtime design includes:

- Point-and-click model configuration and operation;

- Integrated two-dimensional, three-dimensional, geographical information system (GIS), and other model views;

- Automated connections to enterprise data sources; and

- Automated connections to powerful external programs for statistical analysis and visualization of model results.

ANNOTATIONS AND SETTINGS

Repast S uses a new feature in Java 5, namely, annotations, to declaratively mark code for later operations. Annotations are metadata tags that are compiled into binary class files. Like comments, annotations are not directly executed. Unlike comments, annotations are stored in the compiled versions of source code. This storage allows executing Java programs such as the Repast S runtime system to read and act on the encoded metadata. This allows Repast S developers to declaratively mark or annotate code at design time for special processing by the Repast S runtime system. This facility is used for tasks such as declaring "watchers." The example in Figure 1 shows an agent behavior that is activated any time a connected network neighbor or friend changes its "happiness" attribute. Watchers are considered further in North et al. (2005). Annotations are also used for tasks such as scheduling, as shown in Figure 2.

Repast S is expected to use two major types of settings, namely, model and scenario descriptors, to glue or bond models together. Model descriptors define what *can be* in a model, such as the allowed agent types, permitted agent relationships, and watching information. Scenario descriptors define what *actually is* in a model, such as agent data sources, visualizations, and logging. Model and scenario descriptors are stored in XML files. Descriptors are discussed in North et al. (2005).

```
@Watch(watcheeClassName = “repast.user.models.SimpleHappyAgent”,
watcheeFieldName = “happiness”, query = “linked_from”,
whenToTrigger = WatcherTriggerSchedule.LATER, scheduleTriggerDelta = 1,
scheduleTriggerPriority = 0)
public void friendChanged(SimpleHappyAgent friend) {
     if (Math.random() > .25) {
            this.setHappiness(friend.getHappiness());
     } else {
            this.setHappiness(Random.uniform.nextDouble());
     }
}
```

FIGURE 1 An example "Watcher" annotation for a simple happy agent method

```
@ScheduledMethod(start = 1, pick = 1)
   public void changeHappiness() {
      this.happiness = Random.uniform.nextDoubleFromTo(0, 1);
   }
```

FIGURE 2 An example "Scheduler" annotation for a simple happy agent method ("Start" is the time step to call the method, and "Pick" indicates random selection of one of the available simple happy agents)

Model descriptors are to be created at model development time, while scenario descriptors are expected to be created at runtime. The Repast S development environment is expected to provide both a wizard for creating and a point-and-click editor for modifying model descriptors. The Repast S runtime environment includes a point-and-click panel for creating and maintaining scenario descriptors.

THE WEASELS

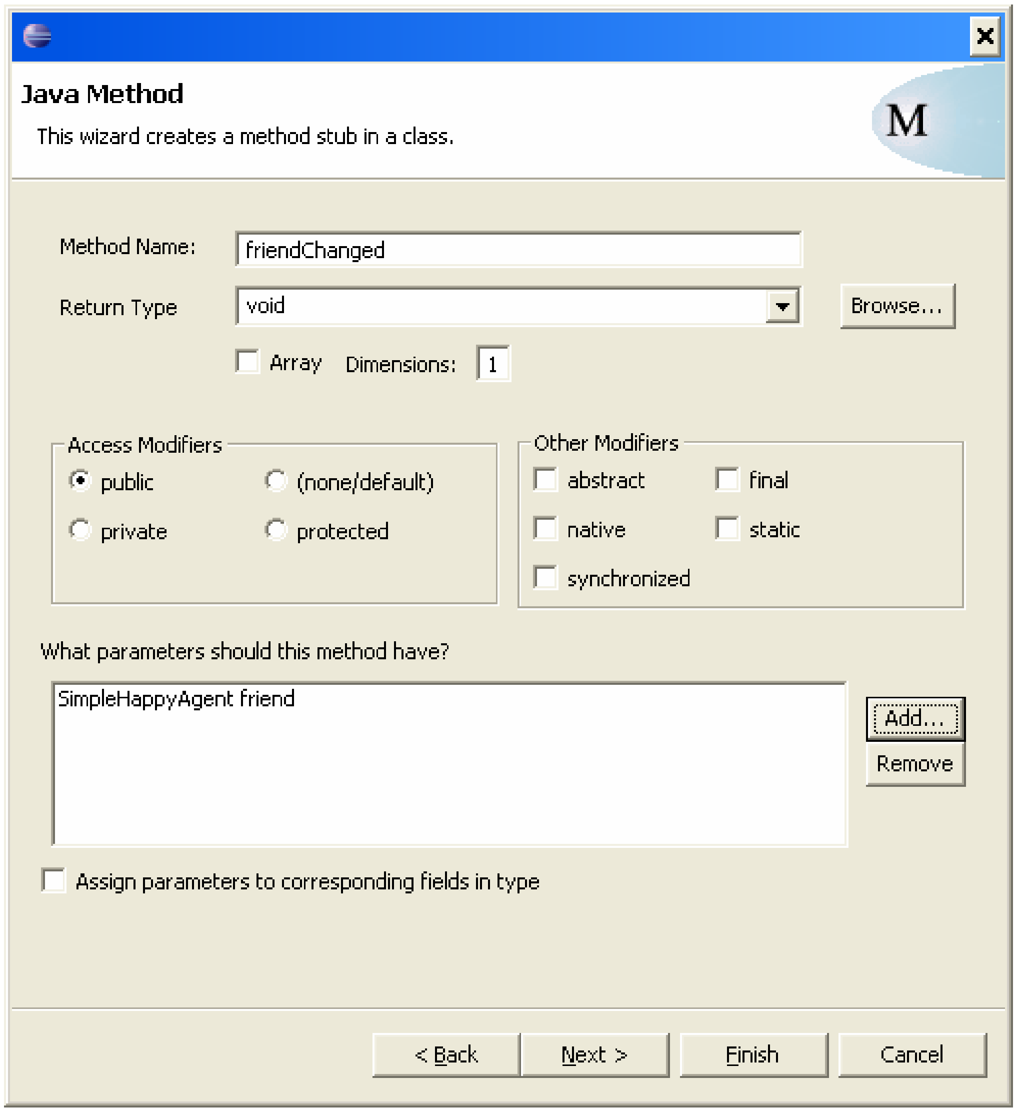
The Repast S development environment is expected to use the CodeWeasel Eclipse plugin system. CodeWeasel is a set of Java plugins for the Eclipse development environment (Eclipse 2005). CodeWeasel is being developed by the Argonne Repast team to support systems such as Repast S. CodeWeasel is expected be released as a separate free and open-source project that supports Repast S.

CodeWeasel is expected to work within Eclipse to automate and simplify the creation and maintenance of Java code. Eclipse itself contains powerful tools to create and maintain Java packages and classes. CodeWeasel contains tools that augment and fill in gaps in these existing functions. The guiding design rule for CodeWeasel is that only pure Java files are used. No *separate* non-Java metadata or state files are allowed to store user code. Currently, the main members of the CodeWeasel family are MethodWeasel, FieldWeasel, and LegacyWeasel. Additional tools may also be introduced.

MethodWeasel has a wizard and visual editor for Java methods. The wizard, shown in Figure 3, provides a point-and-click tool for creating new method signatures (e.g., method name, method parameters, and method return type) and specifying method annotations. The wizard currently builds on the free and open-source Eclipse Plug-in Method Wizard (Hawlitzek 2005). The visual editor represents Java code as an editable flowchart as shown in Figure 4. The contents of the flowchart are planned to have a direct correspondence with Java code so the editor can work with almost any standard Java 5 code, regardless of whether or not it was originally created with MethodWeasel. The visual editor also is expected to provide point-and-click tabs for modifying method signatures and changing method annotations.

Much like MethodWeasel, FieldWeasel has a wizard and visual editor. Unlike MethodWeasel, FieldWeasel works with Java fields. The wizard provides a point-and-click interface for creating and initializing new fields, as shown in Figure 5. As before, the wizard currently builds on the free and open-source Eclipse Plug-in Method Wizard (Hawlitzek 2005). The visual editor is expected to allow point-and-click editing of the field signature and annotations, and is expected to represent the field's initialization code as an editable flowchart.

LegacyWeasel is a model integration tool that is expected to allow developers to use straightforward XML documents to specify the format of legacy files (e.g., existing text and binary files) and the source or destination of the data in Java (e.g., the Java classes that produce or consume the data). These XML files are expected to be creatable using a point-and click editor within Eclipse. Once the appropriate XML files are created, LegacyWeasel is expected to automatically convert the given data sources (e.g., Java objects) into input files; run or activate the legacy model or programs; and then read the resulting output file contents back into the appropriate destinations (e.g., the same or different Java objects). In addition to the goal



Method Name: friendChanged
Return Type: void
Array [ ] Dimensions: 1

Access Modifiers: [✓ public] (none/default) private protected

Other Modifiers: abstract [ ] final [ ] native [ ] static [ ] synchronized [ ]

What parameters should this method have?
SimpleHappyAgent friend

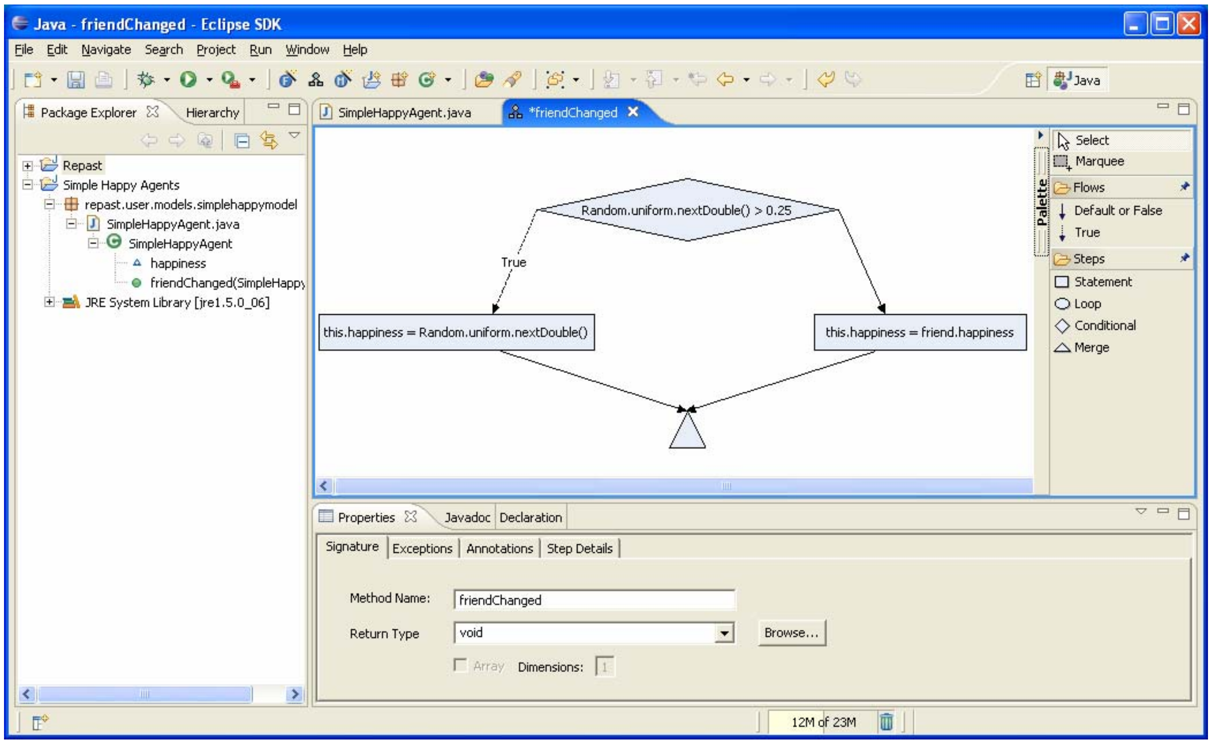
[ ] Assign parameters to corresponding fields in type

FIGURE 3 One of the pages in MethodWeasel's method wizard

of largely automating and greatly simplifying the often tedious model integration process, LegacyWeasel XML files have the potential to be used as detailed documentation on the format and content of legacy model input and output files.

MODEL TOOLS

As previously stated, CodeWeasel is being developed by the Argonne Repast team to support systems such as Repast S. The various tools within CodeWeasel are expected to provide model developers with a range of useful functions. In addition to these general-purpose tools, the Repast S development environment is expected to include a set of specific support tools. These tools are expected to include a new model wizard, a new model specification file wizard, and a



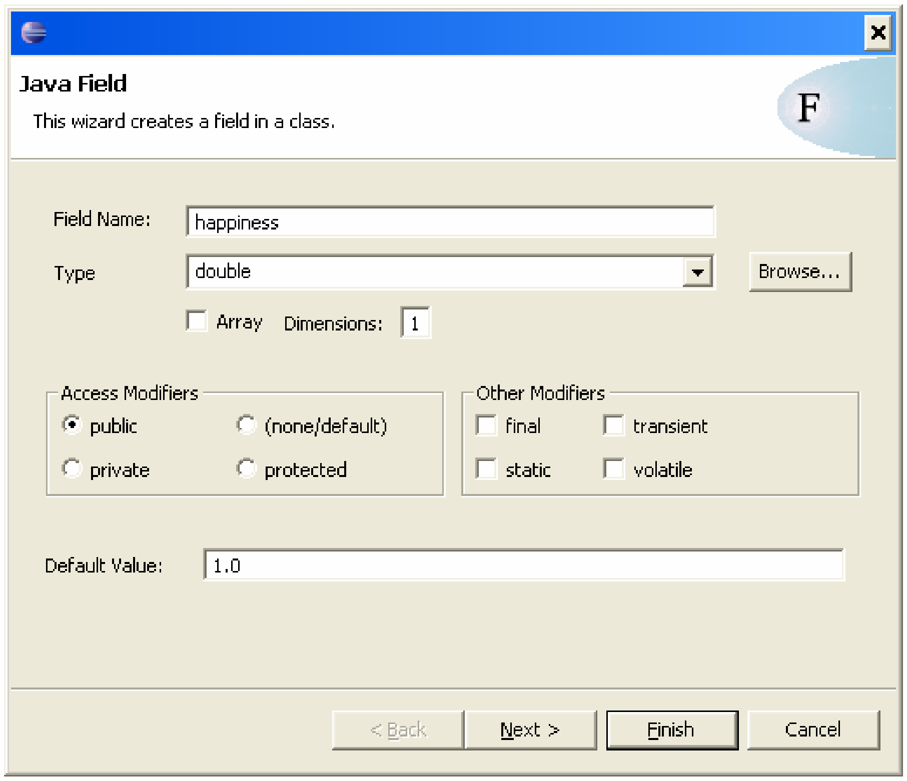
```
Random.uniform.nextDouble() > 0.25
True
this.happiness = Random.uniform.nextDouble()
this.happiness = friend.happiness
```

Method Name: friendChanged

Return Type: void

Array [ ] Dimensions: 1

FIGURE 4 The MethodWeasel method editor



Java Field

This wizard creates a field in a class.

Field Name: happiness

Type: double

Array [ ] Dimensions: 1

Access Modifiers: [✓ public] (none/default) private protected

Other Modifiers: final [ ] transient [ ] static [ ] volatile [ ]

Default Value: 1.0

FIGURE 5 One of the pages in FieldWeasel's field wizard

model specification file editor. The new model wizard is expected to be used to create new Repast S models (i.e., Repast S projects) in Eclipse. The new model specification wizard is expected to be used to create model specification XML files, while the model specification editor is expected to allow these files to be updated on a point-and-click basis. Additional tools may also be provided.

CONCLUSIONS

The Repast S runtime is a pure Java extension of the existing Repast portfolio. Repast S extends the Repast portfolio by offering a new approach to simulation development and execution. The Repast S development environment is expected to include advanced features for agent behavioral specification and dynamic model self-assembly. Any POJO can be a Repast S model component. This paper discusses one powerful agent modeling-focused way to create such POJOs, namely, the Repast S development environment. However, any method from hand coding, to wrapping binary legacy models, to connecting into enterprise information systems can be used to create Repast S model components. Once the model components are created, Repast S is expected to provide a set of point-and-click tools for binding the components into working models.

ACKNOWLEDGMENT

The authors wish to thank David L. Sallach for his visionary leadership in founding the Repast project and Charles M. Macal for sustaining involvement in the project. This work is supported by the U.S. Department of Energy, Office of Science, under contract W-31-109-Eng-38.

REFERENCES

Eclipse, 2005, *eclipse*, Eclipse home page, Eclipse Foundation Inc., Ottawa, Ontario, Canada; available at http://www.eclipse.org.

Hawlitzek Consulting, 2005, *Eclipse Plug-in Method Wizard*; available at http://www.hawlitzek-consulting.de/Java_Downloads/Eclipse_Extensions/Eclipse_Extensions_English/eclipse_extensions_english.html.

North, M.J., T.R. Howe, N.T. Collier, and J.R. Vos, 2005, "Repast Simphony Runtime System," in C.M. Macal, M.J. North, and D. Sallach (eds.), *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, ANL/DIS-06-1, co-sponsored by Argonne National Laboratory and The University of Chicago, Oct. 13–15.

North, M.J., N.T. Collier, and J.R. Vos, 2006, "Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit," *ACM Transactions on Modeling and Computer Simulation* **16**(1):1–25, ACM, New York, NY, Jan.

ROAD, 2005, *Repast*, Repast home page, Repast Organization for Architecture and Design, Chicago, IL; available at http://repast.sourceforge.net.

SDG, 2005, *Welcome to the Swarm Development Group Wiki!*, Swarm home page, Swarm Development Group, Santa Fe, NM; available at http://www.swarm.org/wiki/Main_Page.

Wilensky, U., 1999, *NetLogo*. Center for Connected Learning and Computer-based Modeling, Northwestern University, Evanston, IL.